LibGDX Game Development Essentials

Make the most of game development features powered by LibGDX and create a side-scrolling action game, Thrust Copter

Juwal Bose
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 *Wiring Up*
- A synopsis of the book’s content
- More information on *LibGDX Game Development Essentials*

About the Author

**Juwal Bose** is a game developer, game designer, and technology consultant from the incredibly beautiful state of Kerala in India. He is an active figure in social media and game development SIGs, and he never misses a chance to speak at technical conferences and BarCamps. He conducts technical workshops for engineering students at professional colleges as part of open source initiatives. Juwal is the Director at Csharks Games and Solutions Pvt. Ltd., where he manages research and development, training, and pipeline integration.

He has been developing games since 2004 using different technologies, such as ActionScript, Objective-C, Java, Unity, LibGDX, Cocos2D, OpenFL, Unity, and Starling. His team has created more than 400 games to date, and many of the job management games are listed at the top of leading portals worldwide. He was involved in the development of more than 20 LibGDX games, primarily for the Android platform.

Juwal writes game development tutorials for GameDevTuts+ and manages the blog of Csharks. His isometric tutorial for GameDevTuts+ was well received and is considered a thorough guide for developing tile-based isometric games. This is his second book and he aims to keep writing and sharing his 10 years of game development experience through more books. His first book, *Starling Game Development Essentials, Packt Publishing*, was on another exceptional cross-platform game development framework—Starling.

Juwal is a voracious reader and likes to travel. His future plans also include writing fiction.
Acknowledgments

I would like to thank my parents and my wife, Dr. Nidhina Haridas, for inspiring me to aim higher and try harder. My heartfelt thanks go to Eldhose P. Mathew, CEO of Csharks, for allowing me to devote my professional time to work on this book. I would also thank Anila Vincent, Meeta Rajani, and Madhunikita Chindarkar for supporting me and all the other team members at Packt Publishing for giving me this opportunity. A special thanks to Suryakumar for those brainstorming sessions and Kenney for the open source art.

This book would not have been possible without the expertise of the technical reviewers Tomasz Wojciechowski, Lévêque Michel, Pavel Czempin, and Sudarshan Shetty. Thank you guys.

I would also like to express my deep gratitude to Mario Zechner, without whom such an incredible game framework would not have existed. Mario, you rock. Long live LibGDX!
LibGDX Game Development Essentials

Game development is one of the coolest things to do right now, be it as a hobby or as a full-fledged career option. With the arrival of smartphones, tablets, and other smart devices, games are leading all the app stores worldwide in terms of the number of downloads and revenue. Games have opened up revenue streams worth millions of dollars for not only game companies, but also for indies and single developers. App stores support self-publishing and they are very indie-friendly, which means anyone capable of creating a game can go ahead and publish their game and reap the benefits. This book will help you with just that.

Developing a game is not an easy task, but with the right tools at our disposal it becomes easier to develop a new game. Depending on the type of game, the platform, and prior knowledge of the developer, there are different types of tools that can be used. This book explores LibGDX, a Java-based game development framework that is primarily used to develop 2D Android games. However, LibGDX is also capable of deploying to Windows, Mac, Linux, iOS, Blackberry, and HTML5. Unity, Starling, Cocos2D, Phaser, and OpenFL are some of the alternatives that you need to be aware of.

LibGDX (http://libgdx.badlogicgames.com) is an open source, Java-based game development framework. It provides a setup for rapid prototyping and faster iterations, with the capability to develop and debug on our desktop. It is the brainchild of Mario Zechner, the author of Beginning Android Games. The team of LibGDX now has 15 people (https://github.com/orgs/libgdx/people). LibGDX is the undisputed leader when it comes to creating Android 2D games that are performant and universal.

This book aims to help you get started with game development using LibGDX. In this book, you will create a simple game that is similar to Flappy Bird and learn about the different aspects of game development. This book is bundled with the complete source code of the working game Thrust Copter, along with other sample code relevant to individual chapters. This book will help you to learn about game development, UI creation, and data persistence. Once the game is ready, a chapter is dedicated to convert it to a physics-based game using Box2D. Another chapter introduces basic 3D concepts, tile maps, and other tools of the trade. The final chapters detail how to integrate various third-party features and how to publish the game on various platforms.
The book does not try to teach you everything, but makes you ready to explore LibGDX further and encourages you to explore the source code on your own. I would appreciate it if you would support LibGDX once you are aware of its tremendous potential (http://www.patreon.com/libgdx). Once you are done with this book, I would urge you to read LibGDX Cross-platform Game Development Cookbook and Learning LibGDX Game Development, both by Packt Publishing, to take your knowledge to the next level.

What This Book Covers

Chapter 1, Wiring Up, explains how to set up the LibGDX development environment. We will install Eclipse, Android SDK, and LibGDX via Gradle, which helps us to run our Hello World project.

Chapter 2, Let There Be Graphics!, explains how to add game graphics and discusses the Texture, TextureRegion, Sprite, and Animation classes. Texture packing and SpriteBatch are also introduced in this chapter.

Chapter 3, Thou Shall Not Pass!, explains how to add game controls for the player to interact with the game and discusses how to add sound effects and music loop.

Chapter 4, Bring in the Extras!, explores how to add additional game play elements, such as pickups, GUI, and effects. Bitmap fonts and particle effects are added and their respective tools are also explained in this chapter.

Chapter 5, Scene 2 – the Menu, introduces Scene2D and explains the creation of the menu scene along with a loading scene. In this chapter, you will learn about Stage, Actor, and Nine Patch images. This chapter also deals with how to handle Android back button.

Chapter 6, Physics with Box2D, explains how to convert the completed game logic to one that uses physics simulation with the help of Box2D. In this chapter, you will learn about the basics of Box2D and use it to create an alternate implementation of the Thrust Copter game.

Chapter 7, The Amazing World of 3D, explores the 3D capabilities of LibGDX. With the use of sample code, we will render 3D primitives, import models, and animate them. We will also discuss 3D particle effects and 3D bullet physics in brief.
Chapter 8, Saving Our Data, discusses the methods of file access that will help you to preserve persistent data. We will implement a local leaderboard system with this new knowledge. The TMX tile map is also introduced in this chapter.

Chapter 9, Finishing Our Android Game, adds analytics tracking, Google Ads, Google Play services, leaderboard, and achievements to the game.

Chapter 10, Time to Publish, explains how to publish the game to different platforms and app stores. Overlap2D, a great visual scene designer for LibGDX is also introduced in this chapter.
In this chapter, we will lay the ground work to facilitate our plan of creating a game like Flappy Bird for multiple platforms, including Android, iOS, PC, and the Web. You might wish to jump right in and start designing the game, but that is not the case with LibGDX. There are lots of things to be done before we could start to code. This chapter deals with the setting up of LibGDX for a cross-platform project. This is done on a development environment configured for Android development. We will cover the following topics:

- Installing Java Development Kit (JDK)
- Installing Eclipse for Java
- Installing Android SDK
- Installing Google Web Toolkit (GWT) SDK
- Installing all relevant Eclipse plugins, such as Gradle, Android Development Tool (ADT), Google App Engine (GAE), GWT, and RoboVM
- Using the LibGDX Gradle setup app to create our first LibGDX project
- Exploring LibGDX demos and tests
Getting started

Hope I didn't scare you with all those abbreviations. The majority of what we will do in this chapter will only need to be done once, as it is required to set up your development environment. Once it is done, we can forget about it and focus on creating wonderful games. I will be using a Mac-based development setup throughout this book, but the process is similar for Windows as well. The added benefit that Mac provides is the ability to deploy to iOS as well. In the first part, we will set up a Java development environment that will be very simple for all our Java developers. This can be accomplished by setting up Eclipse, IntelliJ, or NetBeans IDEs, but we will be focusing on Eclipse throughout this book. This does not mean that this way is superior to the others but we've used it just because Eclipse is the most widely used Java IDE and the majority of our potential readers should have worked with Eclipse one way or another.

If you need to use another IDE, please check the LibGDX wiki page at http://libgdx.badlogicgames.com/documentation.html#gettingstarted.

Setting up the Java development environment

Our first step will be to install the Java Development Kit (JDK) if the development PC does not have it. For all of the tools, applications, and SDKs that we will use, there are different versions for Windows and Mac. Your browser will automatically take you to the relevant download in most cases, but be sure to double check that you are indeed downloading the right version. Also, there are different versions for 32-bit and 64-bit processors; make sure that you download the right one for your development PC.

You can download the latest version of JDK from Oracle's site at http://www.oracle.com/technetwork/java/javase/downloads/index.html.
At the time of writing this book, the latest version is JDK 8u5 and the following screenshot shows how the different versions are listed:

### Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:
- Java Developer Newsletter (click the checkbox under Subscription Center > Oracle Technology News)
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine
- JDK MD5 Checksum

Looking for JDK 8 on ARM? JDK 8 for ARM downloads have moved to the JDK 8 for ARM download page.

### Java SE Development Kit 8u5

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

<table>
<thead>
<tr>
<th>Product / File Description</th>
<th>File Size</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux x86</td>
<td>133.58 MB</td>
<td>jdk-8u5-linux-x86.rpm</td>
</tr>
<tr>
<td>Linux x86</td>
<td>152.5 MB</td>
<td>jdk-8u5-linux-x86.tar.gz</td>
</tr>
<tr>
<td>Linux x64</td>
<td>133.87 MB</td>
<td>jdk-8u5-linux-x64.rpm</td>
</tr>
<tr>
<td>Linux x64</td>
<td>151.64 MB</td>
<td>jdk-8u5-linux-x64.tar.gz</td>
</tr>
<tr>
<td>Mac OS X x64</td>
<td>207.7 MB</td>
<td>jdk-8u5-macosx-x64.dmg</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit (SVR4 package)</td>
<td>135.88 MB</td>
<td>jdk-8u5-solaris-sparcv9.tar.Z</td>
</tr>
<tr>
<td>Solaris SPARC 64-bit</td>
<td>95.64 MB</td>
<td>jdk-8u5-solaris-sparcv9.tar.gz</td>
</tr>
<tr>
<td>Solaris x86 (SVR4 package)</td>
<td>135.9 MB</td>
<td>jdk-8u5-solaris-x86.tar.Z</td>
</tr>
<tr>
<td>Solaris x64</td>
<td>93.19 MB</td>
<td>jdk-8u5-solaris-x64.tar.gz</td>
</tr>
<tr>
<td>Windows x86</td>
<td>151.71 MB</td>
<td>jdk-8u5-windows-x86.exe</td>
</tr>
<tr>
<td>Windows x64</td>
<td>155.18 MB</td>
<td>jdk-8u5-windows-x64.exe</td>
</tr>
</tbody>
</table>

Once you download the relevant file, go ahead and install it on your machine. Now, your machine is ready to be used for Java-based application development.
Wiring Up

On a Windows machine, you may need to set the value of the environment variable JAVA_HOME to the installation path of the JDK after installation. We can check this by running the following command in the command prompt:

```
C:\Users\admin>java -version
```

If the system displays the version details as follows, then Java is installed correctly:

```
java version "1.7.0_25"
Java(TM) SE Runtime Environment (build 1.7.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
```

Otherwise, we need to set up the Java environment variable. To find the correct path, go to `C:\Program Files (x86)\Java\`. You should see a folder with the jdk extension in its name. The complete path should be similar to the following one:

```
C:\Program Files (x86)\Java\jdk1.8.0_05
```

Follow the given steps on a 64-bit Windows machine with 32-bit Java installed to set the environment variable:

1. Click on the Windows Start button, right-click on **Computer**, select **Properties**, and open control panel system window.

2. Click on **Advanced system settings** on the left-hand side of the Control Panel window to open the **System Properties** window.

3. Next, click on the **Environment Variables...** button and click on the **New...** button at the top that corresponds to **User variables for <USERNAME>**. A window with the title **New User Variable** will appear.

4. Now, fill in the two text fields. Use **JAVA_HOME** in the **Variable name** field and the JDK path you found out earlier in the **Variable value** field.
Installing Eclipse

Next, we will need to install our IDE of choice: Eclipse. The latest version of Eclipse can be found at http://www.eclipse.org/downloads/. We need to select Eclipse for Java developers from the multitudes of flavors in which it is available. At the time of writing, the latest Eclipse build is **Luna**, as shown in the following screenshot:

Once you have downloaded the relevant IDE, go ahead and install it on your machine. Now, we are all set to write and compile the Java code. It's time to set up the Android development environment.
Setting up the Android development environment

To use our Eclipse IDE for Android development, we need to install two things: the Android SDK and the Android ADT plugin. Android SDK can be downloaded as a compressed archive from http://developer.android.com/sdk/index.html. It will be listed under GET THE SDK FOR AN EXISTING IDE. The archive can be extracted to a location on your hard drive which needs to be linked from within Eclipse. The following screenshot shows the download page where you can see a link to download Eclipse ADT, which is a complete package for Android development.

If you are setting up Eclipse for the first time, then downloading Eclipse ADT is the way to go; however, in this chapter, we are assuming that Eclipse is already installed.

Installing the ADT plugin and linking the Android SDK with Eclipse

The ADT plugin connects with our Android SDK and keeps it up to date. It also has the SDK Manager and Android Virtual Devices that are used to emulate Android devices. We need to fire up Eclipse so that we can install the plugin. For those who are not aware of the process of installing plugins in Eclipse, the following screenshot will help. We need to select Install New Software... from the Help section.
In the new window that pops up, enter the URL https://dl-ssl.google.com/android/eclipse/ in the section that says type or select a site and press Enter. We need to select all available items and proceed with the installation. Once the plugin is successfully installed, we need to restart Eclipse. After restarting, Eclipse will ask you for the Android SDK location; you can also set it up by navigating to Window | Preferences | Android. Once the Android SDK location is set, the SDK Manager will check our installation to find missing items. It will prompt you to download the latest Android SDK platform and Android platform tools. Check out the selected items in the Download dialog box. We need to have the recommended Android build tools and SDK platform. It is safe to not alter the recommended setting and allow the downloader of Eclipse to download all the required files. The following screenshot shows the update in progress:

Now we are all set to develop Android applications.
Installing the LibGDX support plugins

LibGDX uses GWT plugin to publish HTML5/JavaScript, which is the web platform. GWT includes the GWT SDK and Google App Engine. The other support plugins that we need are RoboVM and Gradle.

RoboVM is used to compile the LibGDX project on the iOS platform. Gradle is a dependency management and build system that wires our LibGDX game together.

Setting up GWT

From within Eclipse, launch the Install New Software... window, input the following link, and install the GWT plugin:

https://dl.google.com/eclipse/plugin/4.4

The last part of the link is actually the Eclipse version and the preceding link is for Version 4.4.x. Select the checkboxes to install Google Plugin for Eclipse, Google App Engine SDK, and Google Web Toolkit SDK. Once the plugin is installed, we need to restart Eclipse.

Installing the RoboVM plugin

iOS development is only possible if we are using a Mac machine for development. We will also need Xcode, the Mac IDE for support. RoboVM is the brainchild of Niklas Therning, a Swedish developer and co-founder of Trillian Mobile AB. RoboVM is used to make Java work on iOS via a Java to Objective-C bridge. RoboVM is open source and stable. Let's all take a few minutes to appreciate this wonderful effort by learning more about RoboVM (http://www.robovm.com/).

Mac users can go ahead and install the RoboVM plugin from http://download.robovm.org/eclipse/. After restarting, we are now all set to deploy our games on the Apple iOS platform as well.

Installing the Gradle plugin

Gradle is a dependency management system and is an easy way to pull in third-party libraries into your project without having to store the libraries in your source tree. Instead, the dependency management system relies on a file in your source tree that specifies the names and versions of the libraries you need to be including in your application.
Adding, removing, and changing the version of a third-party library is as easy as changing a few lines in that configuration file. The dependency management system will pull in the libraries you specified from a central repository and store them in a directory outside of your project.

Gradle also has a build system that helps with building and packaging your application, without being tied to a specific IDE. This is especially useful if you use a build or continuous integration server, where IDEs aren't readily available. Instead, the build server can call the build system, providing it with a build configuration so it knows how to build your application for different platforms. More information can be found at http://www.gradle.org/.

At this point, we are very comfortable adding new plugins to Eclipse. Fire up the new software window to add the Gradle plugin from the link http://dist.springsource.com/release/TOOLS/gradle. After installation, restart Eclipse and that's the end of our setup.

The LibGDX Gradle combo

I know that you are wondering how we reached the end of our setup without installing anything related to LibGDX but everything else out there. The LibGDX installation will be handled by Gradle automatically, and we will be using a helper application to create all the dependencies and the project structure. Such a LibGDX Gradle combo requires an Internet connection while creating the project, as Gradle needs to load all the necessary dependencies, files, and libraries on the fly while setting up the project for the first time. Personally, I am not a fan of this as I come from a country where a good Internet connection is still a luxury. For those of you who are in a similar situation, the alternative way will be to download all dependencies and wire them all up as required. This is a complicated task although we used to do it during the initial days of LibGDX.

Let's start creating our first Gradle-based LibGDX project. We will be following the steps explained in the LibGDX wiki page, which can be found at https://github.com/libgdx/libgdx/wiki/Project-Setup-Gradle.
Using the Gradle setup application

We need to use a Java application with the Gradle setup to create Gradle-based LibGDX projects. This setup file can be found at http://libgdx.badlogicgames.com/download.html.

A direct link to the setup application is http://bitly.com/1i3C7i3.

The setup application has to be stored for easy access, as we will need it when we create new LibGDX projects. The purpose of the application is to create all platform-specific Eclipse projects, such as desktop, Android, iOS, and JavaScript applications. It also links with all the necessary dependency libraries and the latest version of LibGDX. The following screenshot shows the Gradle project structure that clearly explains how the different projects are created under the relevant folders:

<table>
<thead>
<tr>
<th>Project layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>settings.gradle</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>gradlew</td>
</tr>
<tr>
<td>gradlew.bat</td>
</tr>
<tr>
<td>gradle</td>
</tr>
<tr>
<td>local.properties</td>
</tr>
<tr>
<td>core/</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>src/</td>
</tr>
<tr>
<td>desktop/</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>src/</td>
</tr>
<tr>
<td>android/</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>AndroidManifest.xml</td>
</tr>
<tr>
<td>assets/</td>
</tr>
<tr>
<td>res/</td>
</tr>
<tr>
<td>src/</td>
</tr>
<tr>
<td>gwt/</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>src/</td>
</tr>
<tr>
<td>webapp/</td>
</tr>
<tr>
<td>ios/</td>
</tr>
<tr>
<td>build.gradle</td>
</tr>
<tr>
<td>src/</td>
</tr>
</tbody>
</table>
A Hello World project
Let's launch the gdx-setup.jar Gradle setup application. The following screenshot shows how I have populated the options for our Hello World project:
Wiring Up

We need to specify a name for our project, which is Hello World in this case. Then, we need to specify a package name for our project. In my case, I am using my company's package name but you can use any unique package name. It's time to specify our main class name; take care that there are no spaces in between. While setting the destination project, make a new folder within your Eclipse Workspace folder so that there are no possible conflicts between Gradle files and Eclipse workspace metadata files. In this case, I have specified FirstGradleProject. Link the Android SDK and select the version of LibGDX that you will use.

In the Sub Projects section, we need to select the platforms that we want the project to target. Note that we will need a Mac to compile an iOS build.

The Extensions section will link any of the standard LibGDX libraries or packages available. It is always safe to select the options in this stage rather than hacking into the project later on. For our Hello World app, we do not need any of these extensions. Hit the Generate button and your LibGDX Gradle project structure will be created.

Alternatively, we can create an Eclipse-specific project structure by clicking on the Advanced button, enabling the Eclipse checkbox, and clicking on Save. If the project is created this way, then we need to import it as a normal Eclipse project by navigating to File | Import | General | Existing Projects to Workspace. This process will need an active Internet connection to load the files needed by the setup application.

Importing Gradle projects to Eclipse

It's time to import our projects into Eclipse by navigating to File | Import | Gradle | Gradle Project. Browse the root folder, FirstGradleProject, and hit the Build Model button. This step is very important, as you may not see your projects without this. Gradle will download a few necessary files based on our project, and it will take a while before you see your projects listed as shown in the following screenshot:
Go ahead and select all the projects and click on **Finish** to load them to Eclipse. At this point Gradle will load all the dependencies as per our selection in the setup application. This is where Gradle will actually load the LibGDX packages and extensions. The following screenshot shows the process:

![Image of Gradle loading process]

---

**Running the project**

Everything should be wired properly and all projects are good to go at this point. One issue that may pop up is that the Android project may have a red cross indicating that the proper Android SDK is missing. This happens all the time, but the fix is straightforward. Right-click on the Android Project folder to select its properties. Then, select the proper Android target version under the **Android** section in the window that pops up. This will remove the error. Congratulations! You can now run your projects. The following screenshot shows how to remove the Android error:

![Image of removing Android error]
In order to run the desktop project, right-click on the Hello World desktop project, select Run As, and click on Java Application. A popup may ask you to select the application class and you should select DesktopLauncher to run the app. The following window will pop up, which means we have successfully created and run our first Gradle-based LibGDX application:
Running the application on your Android device is also very easy. Connect your Android device to the development PC via USB and make sure USB debugging is enabled in the device's settings.

USB debugging can be enabled by navigating to Settings | Developer options | USB debugging.

In order to see the connected device from within Eclipse, you need to enable the Devices view. In Eclipse, go to Window | Show View | Other. Then, select Devices from the Android section. A new tab showing the connected device will be added to Eclipse. For Windows, we need to install the respective drivers for the connected phone to show up, but on Mac the connected device would be automatically detected.

Once you see your device listed in the Devices view, right-click on the Hello World-android project, select Run as, and click on Android Application. Eclipse will prompt you to select how to launch the Android app. You can select to launch on a device or on a Android Virtual Device (AVD) if you have set one up already. Select to launch the application on the device and then the application should show up on your Android device. Eclipse will show the LogCat view, which shows the application status:

In order to run the iOS project, you need to be on a Mac and should have Xcode Version 5 or above installed. Xcode can be downloaded free from the Mac App Store. It provides the necessary frameworks and the iOS Simulator tool. Right-click the Hello World-ios project, select Run As, and select iOS Simulator App.
You can select either of the simulator options, iPad or iPhone. This will start the RoboVM cross compilation and would take some time for entire classes to be compiled. Eventually, you will see the app running on the iOS Simulator.
In some cases, you may run out of memory while RoboVM works on the compiling. You may need to increase the memory heap sizes for Eclipse in the `eclipse.ini` file, as follows:

- `-Xms512m`
- `-Xmx2048m`

Getting the HTML project to run is the trickiest part of them all. We need to follow exactly what the LibGDX wiki tell us to do. Right-click on the `Hello World-html` project, select **Run As**, and click on **External Tools Configuration**. Create a new configuration by double-clicking the **Program** entry in the left sidebar. Give the configuration a name, for example, **GWT SuperDev**. Set the location field to the `gradlew.bat` (Windows) or `gradlew` (Linux, Mac) file. Set the working directory to the root folder of your project and specify `html:superDev` as the argument. Click on **Apply** and then click on **Run**.

Wait until you see the message **The code server is ready in the console view**. After that, go to the URL `http://localhost:8080/html`. You can leave the server running. If you change the code, simply click on the **SuperDev Refresh** button in the browser. This will recompile your app and reload the site.

Check out this screenshot to see the app running on a browser:
Exploring the project

That was some heavy lifting; now we can take a break to explore the project in detail. You are seeing an image in the bottom-left corner of the app window. Let's see how we got this output. The core code logic will always be present in the project's src folder, which is the Hello World-core src folder in our case. We can find the HelloWorld.java main class inside the specified package; open it up. This class extends the ApplicationAdapter class, which in turn implements the ApplicationListener interface. These classes have a few methods that align with the Android application life cycle methods: create, pause, resume, dispose, render, and resize.

The HelloWorld class is a very simple class with only two variables and two overridden functions:

```java
SpriteBatch batch;
Texture img;
create();
render();
```

The SpriteBatch class is a class that facilitates the efficient drawing of images on the screen. The create function, which will be called when the application is launched, just creates a new SpriteBatch class and a Texture class named img that loads a texture from the external asset badlogic.jpg. The render function is called continuously and can be considered as our game loop. It clears the screen and draws the texture onto screen at the coordinates (0,0). This means the origin of the coordinate system in LibGDX is the bottom-left corner of the screen, as we can see that the image is placed there. We will revisit the graphics package in the next chapter.

The following code clears the screen before each draw call:

```java
Gdx.gl.glClearColor(1, 0, 0, 1);
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```

The next bit of code starts the batch drawing, draws the texture at the specified coordinates, and closes the batch drawing:

```java
batch.begin();
batch.draw(img, 0, 0);
batch.end();
```
The only thing missing is the external image file, as we won't be able to find it among the core project files. LibGDX follows the convention to store all asset files within the assets folder in the Android project. Hence, we can find badlogic.jpg within the assets folder in the Hello World-android project. The assets folder is shared among all the other projects. All the other projects are simple wrapper projects that have platform-specific code and data that will launch the code in the core project. You can easily explore the projects on your own, but we will revisit specific platforms in the final chapter.

Displaying the Hello World text

Let's alter the code to display our Hello World text thereby officially declaring the start of our quest of mastering LibGDX. The easiest way to do this will be to draw some text on the screen using the BitmapFont class. A bitmap font is created using an external tool like Hiero (https://github.com/libgdx/libgdx/wiki/Hiero) where a font is converted into a bitmap with all the letters in a fixed size along with a .fnt file that stores the presentation data. To make things easier, let's use fonts from the LibGDX test project. You can download verdana39.fnt and verdana39.png from https://github.com/libgdx/libgdx/tree/master/tests/gdx-tests-android/assets/data. As explained, we need to place these files in the assets folder within the Hello World-android project.

Now, let's edit the code in HelloWorld.java in the core project to remove the texture drawing and to add our text display. The code is as follows:

```java
public class HelloWorld extends ApplicationAdapter {
    SpriteBatch batch;
    BitmapFont font;

    @Override
    public void create () {
        batch = new SpriteBatch();
        font = new BitmapFont(Gdx.files.internal("verdana39.fnt"),
                               Gdx.files.internal("verdana39.png"), false);
        font.setColor(Color.RED);
    }

    @Override
    public void render(float delta) {
        batch.begin();
        font.begin(batch);
        font.setColor(Color.RED);
        font.draw(batch, "Hello World", 100, 300);
        font.end();
        batch.end();
    }
}
```
public void render ()
{
    Gdx.gl.glClearColor(0, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    batch.begin();
    font.draw(batch, "Hello World", 200, 200);
    batch.end();
}

@Override
public void dispose()
{
    batch.dispose();
    font.dispose();
}

Running demos

For the adventurous among you, this is the chance to dive deeper into LibGDX by checking out the fully functional game demos. You can find the links for these games at https://github.com/libgdx/libgdx/wiki/Running-Demos. The most interesting one is the example project similar to Flappy Bird, The plane that couldn't fly good. The link to this project is https://github.com/badlogic/theplanetthatcouldntflygood. In order to explore this project, we need to clone the repository in Git or download it on our PC. Once downloaded, we can import it to Eclipse as a Gradle project.
The `local.properties` file may be missing from the downloaded project's folder. Just copy the file from our `FirstGradleProject` root folder. This file sets the Android SDK location in our PC.

This is a very simple project to get you started and can be used as a starter code for your Flappy Bird game. The whole code is in the single file `PlaneGame.java` in the core project.

**Running tests**

The LibGDX tests project is a treasure trove of goodies. There are numerous small tests exploring the different features of LibGDX. These tests are the ideal way to get started with LibGDX and they also let you learn new things in an easy way. Alternatively, they act as a source of functional sample code that we can simply copy and paste for our own use cases. The LibGDX wiki entry for running these tests advises us to use Ant to set up these tests on our PC. The link is https://github.com/libgdx/libgdx/wiki/Running-Tests. The tests can be found at https://github.com/libgdx/libgdx/tree/master/tests/gdx-tests/src/com/badlogic/gdx/tests.

You can temporarily set up Ant after downloading it from http://ant.apache.org/bindownload.cgi. Assuming that it is extracted to `/Setups/apache-ant-1.9.4`, we can temporarily install ANT using the terminal on a Mac. In the terminal, enter the following code:

```
export ANT_HOME=/Setups/apache-ant-1.9.4
export PATH=${PATH}:${ANT_HOME}/bin
```

In the terminal, navigate to the folder where LibGDX Git project is cloned. Run the Ant command to fetch dependencies:

```
ant -f fetch.xml
```
Wiring Up

This will download all the LibGDX dependencies for all the projects. Later, we can import all these projects into Eclipse by navigating to Import | Existing Projects into Workspace. After importing the Demo and Tests projects, our Package Explorer in Eclipse will look something like this:
The test examples are contained in the gdx-tests project. This project only contains the source code. To actually start the tests on the desktop, you have to run the LwjglTestStarter class contained in the gdx-tests-lwjgl project. To run the tests on Android, simply fire up the gdx-tests-android project in the emulator or on a connected device! To run the tests in your browser, fire up the gdx-tests-gwt project. For iOS, you can start the gdx-tests-robovm project. Play with them.

The alternate LibGDX setup

Previously, LibGDX had to be manually wired up and was a complicated process. Later, the French developer Aurelien Ribon created a gdx-setup-ui tool that automated this process. Aurelien Ribon is a rock star Java developer who has released lots of goodies that can be accessed on his blog at http://www.aurelienribon.com/blog/.

This non-Gradle-based system still works but is not recommended or supported at present. The blog post with all the details of Version 3.0.0 can be found at http://www.aurelienribon.com/blog/2012/09/libgdx-project-setup-v3-0-0/. The gdx-setup-ui interface can be downloaded from https://github.com/libgdx/libgdx-old-setup-ui.

The app will need an Internet connection to download the LibGDX files; this screenshot shows it in action:
Wiring Up

Summary
This chapter explained a lot of theory that lays the solid foundation for all our development. This chapter will serve as a reference to set up your development environment if you move to a new workplace or get a new laptop. You learned to set up Eclipse-based Android and Java development environments. We used the gdx-setup tool to create a Gradle-based LibGDX project, which was successfully imported to Eclipse with the help of associated support plugins. We explored the project structure of a typical LibGDX cross-platform project.

We successfully executed the Hello World project on desktop, Android, iOS, and the browser. Some minor editing of the code helped us display the text Hello World using the BitmapFont class. It is very important to successfully import the LibGDX Demo and Tests and we used Ant to get them running.

In the next chapter, we will start with the graphics package in LibGDX and get started with our game.
Where to buy this book

You can buy LibGDX Game Development Essentials from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.